International Journal of Neural Systems, Vol. 31, No. 9 (2021) 2150035 (17 pages)
© World Scientific Publishing Company
DOI: 10.1142/S0129065721500350

A Multi-Objective Evolutionary Approach Based on Graph-in-Graph for Neural Architecture Search of Convolutional Neural Networks

Yu Xue
School of Computer and Software
Nanjing University of Information Science and Technology
Nanjing, P. R. China
Engineering Research Center of Digital Forensics
Ministry of Education
Nanjing University of Information Science and Technology
Nanjing, P. R. China
xueyu@nuist.edu.cn

Pengcheng Jiang
School of Computer and Software
Nanjing University of Information Science and Technology
Nanjing, P. R. China
pcjiang@nuist.edu.cn

Ferrante Neri*
COL Laboratory, School of Computer Science
University of Nottingham, Nottingham, UK
ferrante.neri@nottingham.ac.uk

Jiayu Liang
Tianjin Key Laboratory of Autonomous Intelligent Technology and System
Tiangong University, Tianjin, P. .R. China
yyliang2012@hotmail.com

Received 30 May 2021
Accepted 31 May 2021
Published Online 24 July 2021

With the development of deep learning, the design of an appropriate network structure becomes fundamental. In recent years, the successful practice of Neural Architecture Search (NAS) has indicated that an automated design of the network structure can efficiently replace the design performed by human experts. Most NAS algorithms make the assumption that the overall structure of the network is linear and focus solely on accuracy to assess the performance of candidate networks. This paper introduces a novel NAS algorithm based on a multi-objective modeling of the network design problem to design accurate Convolutional Neural Networks (CNNs) with a small structure. The proposed algorithm makes use of a graph-based representation of the solutions which enables a high flexibility in the automatic design. Furthermore, the proposed algorithm includes novel ad-hoc crossover and mutation operators. We also propose a mechanism to accelerate the evaluation of the candidate solutions. Experimental results demonstrate that the proposed NAS approach can design accurate neural networks with limited size.

Keywords: Deep learning; neural architecture search; multi-objective optimization; genetic algorithm.

*Corresponding author.

1. Introduction

Convolutional neural networks (CNNs) have achieved remarkable results in solving many problems, such as image classification[16] and image segmentation.[40] CNNs are very efficient at obtaining features from 1D sequences of data, 2D images, and 3D images. The features extracted from 1D sequences of sound data by 1D CNNs can be used to extract voiceprint features.[48,96] The features extracted from 2D image data by CNN can be used for image content recognition, prediction, and segmentation. The features in 3D space in 3D image data (mostly medical image data) can be extracted by 3D convolution kernels, which is very useful in predicting diseases and identifying lesions.[34,35] In addition, video data with time attributes can also be classified by 3D CNNs.[21,73]

Among the plethora of real-world applications of CNNs, some modern examples representing the state-of-the-art in the field of neural systems are to analyze the electroencephalogram signals to diagnose seizures[2,42,46] or depression.[3] A neural system based on multiple CNNs is proposed in Ref. 52 to control epileptic seizures. Other studies propose CNNs to diagnose epilepsy in infants[7] and children[43] by classifying electroencephalogram signals. CNNs have been also successfully used to classify medical images to diagnose Parkinson's disease[12,54] and detect pupils.[72] Another popular application domain for CNNs is civil engineering. Some examples of application include damage detection in concrete structures[39] and roads.[53] Some other examples are about vibration-based structural state identification[95] and effect of wind on structures.[59] In addition, CNNs can be combined with other technologies to be applied in more fields. In Ref. 56, CNNs are combined with Long Short-Term Memory to accurately predict the remaining useful life of components, thus helping to make an optimal decision for maintenance management.

There have been many classical network structures, such as AlexNet,[36] VGG,[74] GoogLeNet,[84] Inception-V4,[83] Inception-ResNet,[83] ResNet,[31] DenseNet,[33] etc., which appear to perform well in image classification and image segmentation. However, due to high complexity, it is impractical to use these CNNs on mobile platforms since they would require an excessive amount of computational resources thus leading to an unreasonable waiting time, memory overflow, and high-energy consumption. Therefore, some new lightweight network structures for mobile platforms have been proposed, such as MobileNet,[68] ShuffelNet,[51] MnasNet,[85] EfficientNet,[86] Xception,[19] etc. All the network structures mentioned above are the result of (human) expert design.

In recent years, Neural Architecture Search (NAS) methods,[17] that automatically search the network architectures, are progressively becoming more popular to design CNNs. Most NAS methods are to search the blocks or cells which are consist of convolution kernels with different sizes (such as 3×3, 5×5, etc.) and the position of pool layers.[50,80,82] Moreover, in MUXConv[49] and ShuffleNet,[51] it is pointed out that the generalization performance of the network can be improved by channel multiplexing, spatial multiplexing, and channel shuffling, and then the accuracy of recognition can be improved. The majority of the NAS methods in the literature perform the automatic design by using accuracy as the sole objective of the targets. However, operational efficiency is also an extremely important aspect of the functioning of the network, especially in mobile applications.

In order to simultaneously address accuracy and computational cost, unlike the other studies in the literature, we propose an encoding mechanism with multi-objective evaluate mechanism of the problem where besides the accuracy of the CNN also the number of network parameters is taken into consideration.[65,66,69,70,75,88]

Existing NAS methods design and limit the search space and search domain to reduce the time complexity of the optimization problem. An usual strategy consists of defining some building blocks which are defined by a human expert. This study proposes a graph-based flexible representation that supports a higher level of automatism of the design process. Furthermore, the proposed method relates to the concept of regularized evolutionary algorithm[62,67] in that the approaches aim at reducing the computational overhead (e.g. memory employment) by performing an action on the optimization algorithm.

The remainder of this paper is organized in the following way. Section 2 provides the background about NAS methods, encoding mechanism and evaluation of candidate network architectures. Section 3 provides the details of the proposed NAS method. Section 4 provides the numerical results of this study.

2. Related Work: Neural Architecture Search

The majority of NAS methods can be categorized according to their search logic:

- Gradient-based methods[15,47,92];
- Reinforcement Learning (RL)[9,28,98];
- Evolutionary Algorithms (EAs).[50,63,79,82,91]

This list does not mean to be exhaustive since other methods not belonging to any of the categories above exist, such as Monte Carlo Tree search.[90] The various NAS methods belonging to each category above present advantages and disadvantages. Specifically, RL-based algorithms require a large computational time to perform the automatic design, even on median-scale datasets, such as Cifar-10 and Cifar-100.[37] Unlike RL-based algorithms, gradient-based algorithms are usually very fast. Besides, their search logic leads to obtaining a local optimum problem which may have a much poorer performance than the desired optimal design. Moreover, the gradient-based search algorithm needs to construct a supernetwork in advance, which should contain as much search space as possible. The construction of this supernetwork requires substantial human intervention of an expert, see Refs. 15 and 25. Although EAs are not theoretically guaranteed to converge to the global optimum of problem, they are able to overcome the local optima. Also, they do not require a supernetwork. Thus, EAs are often considered a viable compromise for NAS since they are relatively fast and can be applied to NAS without human intervention or prior knowledge of the problem. One pioneering example is in Ref. 94. It is worthwhile remarking that there exist other search strategies integrated in NAS methods such as Refs. 18, 55, and 57.

This paper focuses on EAs for NAS. In the following sections, some context is provided around the two major challenges of this approach: encoding mechanism and evaluation of the candidate solutions.

2.1. *Encoding of NAS*

The encoding of candidate network architectures for NAS methods are broadly divided into two categories[38]: direct encoding and indirect encoding. Indirect encoding was often used in early works on NAS usually referred to as *Neuroevolution*, see Ref. 71, which is similar to NAS. Neuroevolution uses evolutionary computation to optimize the structure and parameters of neural networks at the same times,[1,4,26,27,30] and many researchers still work on it.[8,32,64,76,77] However, due to the limitations of equipment at that time, the neuroevolution can only be performed on small networks. Furthermore, due to the very large number of parameters in fully connected networks, direct encoding cannot be used to represent the whole network. Therefore, a lot of effort is made to find simple ways (i.e. indirect encoding) to represent the connections and weight parameters of neurons. Thus, indirect encoding is a popular strategy to simplify the search space. These search purposes determine that search space is difficult to represent with direct encoding, so indirect encoding is needed to simplify the encoding and early researchers used indirect encoding to represent individuals.

In recent years, most of the NAS studies have been conducted on neural networks that albeit complex, can be naturally schematized as interconnected blocks. This is the case, besides the CNNs, of Generative Adversarial Networks (GANs),[28] and Recurrent Neural Networks (RNNs).[47] For networks of these types, direct encoding is an easy and natural option. For example, CNNs contain convolution blocks, pooling blocks, batch normalization operations, and sometimes activation functions. These blocks are often represented by a few parameters. Convolution blocks can be fully represented by the number of convolution cores, the size of the convolution cores, stride, padding, dilation, and groups (in fact, some parameters can be directly ignored based on the actual search strategy and purpose). In most cases, pooling blocks, batch normalization operations, and activation functions do not even require parameters for special representations, and they just need the position in the structure to represent the modules.

For each block's position in the structure, there exist two encoding mechanisms:

- linear structure,[81] that is the sequential (linear) arrangement of all blocks or units composed of blocks;
- graph structure,[91] that is a planar (graph) arrangement of interconnected blocks.

Although formally a linear structure is a special graph structure (a sequence is a special graph), we emphasize the distinction since the two encoding mechanisms correspond to two significantly different implementations.

Adjacency matrices are better suited for dense graph structures,[58] since sparse structures can waste a lot of space in adjacency matrices. Sparse structures are better represented by adjacency tables (or adjacency lists). While adjacency matrices are matrices of "0" and "1" to indicate connection or without connection between nodes, adjacency tables are lists that indicate the for each node which nodes are linked to it. The latter allows a compact representation of large sparse networks.

The main advantage of a linear structure is its simplicity compared to that of graph structure. Besides, linear structures cannot represent all the networks. In some cases, like the example in Fig. 1, a linear structure would yield an ambiguous representation of a neural network.

2.2. *Evaluation of NAS*

To evaluate a candidate structure, the general practice is to train the network and calculate its accuracy, see Ref. 80.

Since the training time of the network is very time-consuming, there are many ways to reduce the total time of the evaluation phase. There are two ways to reduce the total time: foresight and early closure. Foresight methods make use of models to predict the performance of the training network. Some researchers use the performance during training to predict the future performance. For example, MetaQNN[10] gives the first 25% of the historical data of the Stochastic Gradient Descent (SGD)

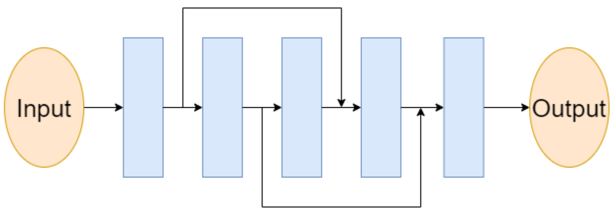


Fig. 1. (Color online) An example of architecture that cannot be represented by a linear structure. Blue blocks are modules in CNNs. This architecture has two skip connections, so it cannot be represented by a linear structure.

training curve to the time series model for prediction and estimates the final accuracy of the network structure. Some researchers use other models, such as random forest, Bayes methods or other models to predict the possible representations of particular network architectures. The reason why they use this method is that the structures searched for by the same NAS method often have a great deal of similarity, and when encoded, it is possible to work out whether the network is good or not from the encoding directly. For instance, PNAS[45] uses the model to predict the top-1 accuracy of candidate networks. Reference 78 proposes an end-to-end offline performance predictor based on the random forest to accelerate the evaluation.

Early closure is another way to reduce the total time of the evaluation phase. This type of approach reduces overall time through targeted evaluations. For example, many researchers used subsets of the dataset for training,[89,99] so that the time of training each network will decrease. Also, Ref. 89 uses a strategy to identify the required structure in advance. In ChamNet,[20] only 300 high-accuracy (or other indicators) samples with different efficiency are selected for each training. Another approach is to keep the good structure and weight so that the new structure requires fewer times to train. There are three specific implementations of this approach: weight sharing, One-Shot method, and weight inheritance. The weight sharing method, which is mostly used in NAS based on gradient, makes use of shared weights from a supernetwork to accelerate the training process, see Refs. 15, 47, 50, 92 and 98. The one-shot method consists of adding components to a small network or deleting components from a large network.[11,22,29,41] The weight inheritance method is mostly used in NAS based on EAs.[14,23,24,63] This method requires that the candidate networks of the entire search space have similar structures. Most of the network structures found by NAS based on EAs meet this condition.

Figure 2 illustrates the weight inheritance method. In the upper part of the figure, two parent solutions with a crossover point (indicated as a diamond) are depicted. The first parent solution is composed of the sequences G_1 and G_2 (representing the network structure) with the corresponding weights W_1 and W_2. Analogously, the second parent solution is composed of G_3 and $G4$ with the weights W_3

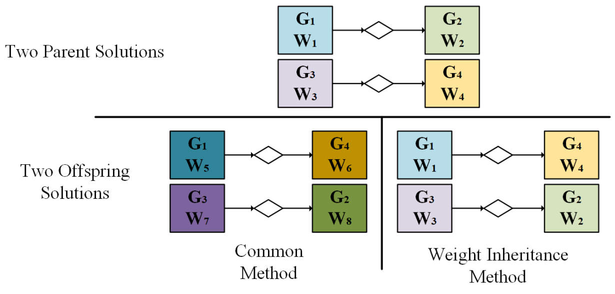


Fig. 2. Comparison between basic crossover (with random initialization of the weights) and crossover with weight inheritance method.

and W_4. In the lower-left part of the figure, the standard crossover is illustrated. The sequences G_2 and G_4 are swapped over and four sets of corresponding weights W_5, W_6, W_7, and W_8 are randomly initialized, thus generating new networks (indicated with a darker color). In the lower-right part of the figure, the weight inheritance method is illustrated. When the crossover occurs, the offspring solutions inherit the weights of the parent (the weights of that portion of the network). Thus, the first offspring solution is composed G_1 and G_4 with the weights W_1 and W_4 while the second solution is composed of G_3 and G_2 with the weights W_3 and W_2.

3. The Proposed Approach: MOGIG-Net

In this section, we introduce the framework of the proposed NAS algorithm, namely Multi-Objective Graph-In-Graph Network (MOGIG-Net) whose flowchart is shown in Fig. 3.

This section first introduces the overall framework of the proposed algorithm and then describes the encoding mechanism, crossover, mutation, decoding method, evaluation, and environment selection in details.

3.1. *Overall description of the MOGIG-Net framework*

Figure 4 displays the structure of the whole algorithm. First, the initial population is obtained through random initialization (line 1), and then the fitness evaluation of the initial population is calculated (lines 2 and 3).

After the initialization, the algorithm makes use of generation cycles to process the population (lines 4–15). New individuals are generated through

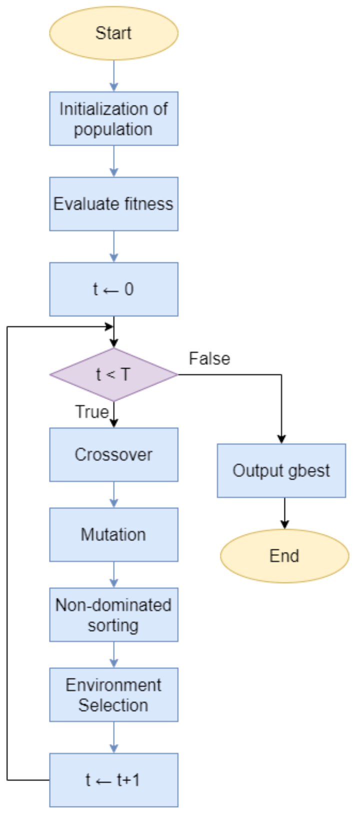


Fig. 3. Flowchart of the MOGIG-Net framework.

Input: The population size p, the maximal generation number T, the crossover probability μ, the mutation probability ν, the maximal nodes number in each block (M_{min}, M_{max}), the maximal blocks number (N_{min}, N_{max}).
Output: Collection of individuals on the pareto frontier meeting the minimization goals.

```
1:  $P_0$ ←Initialize a population with the size of $p$ by using the proposed
    encoding strategy in Algorithm 6;
2:  Convert all genes in $P_0$ to models and evaluate the fitness of the mod-
    els by method 12, and record the fitness of each corresponding indi-
    vidual;
3:  Record the fingerprint and fitness value of each individual.
4:  $t \leftarrow 0$
5:  while $t < T$ do
6:    $Q \leftarrow \phi$
7:    if the length of $Q < P$ then
8:      Two individuals were randomly selected and will cross and mutate
        by the method of Algorithm 7 and 8, and then two offspring will be
        generated.
9:      Record the fingerprint of each offspring, and add the two offspring
        into population $Q$.
10:   end if
11:   $P_t \leftarrow P_t \cup Q$
12:   Convert genes to models and evaluate the fitness of individuals in
      $Q$ by method 12 to control sequence, and record the fitness of each
      individual;
13:   Sort $P_t$ by non-dominated sorting algorithm, and retain $P$ individ-
      uals with better performance, and delete other individuals with poor
      performance, then we will get $P_{t+1}$;
14:   $t \leftarrow t + 1$
15: end while
    Return: $P_{T-1}$
```

Fig. 4. Framework of the MOGIG-Net algorithm.

crossover and mutation. The new individuals are selected for the survival of the fittest by evaluating the fitness values for each objective (lines 12 and 13). Finally, individual sets with better performance on multiple objectives are obtained.

In NAS problems, the evaluation phase is by far the computationally most expensive as it requires the training of the candidate network structure. In order to avoid the reevaluation of the same architectures/structures, we keep an archive of visited solutions with their objective function value. If a solution is revisited the archived objective function values are used.

3.2. *Encoding mechanism of MOGIG-Net*

In this study, we use a graph structure to encode the architecture of the network. We propose the encoding of a CNN in a chromosome divided into blocks linked by separators. To understand the proposed encoding, let us remark that CNNs are composed of blocks, three of them being essential and named (1) convolution; (2) pooling; and (3) fully connection. The chromosome representing the CNN is described as follows:

$$\mathrm{CB}_1 - \mathrm{CB}_2 - \cdots - \mathrm{CB}_n - S - P,$$

where each CB_j is a convolution block, S represents the structure how the convolution block are interlinked, and P describes the presence of pooling layers in the CNN.

The convolution block CB_j is a sequence of separators and binary numbers. The "1" indicates a link between neurons while "0" indicates the dismiss a connection. A convolution block containing m neurons is represented by a sequence of $\frac{m(m-1)}{2}$ binary numbers grouped in sub-blocks of $1, 2, \ldots, m-1$ binary numbers. Each sub-block is separated by a dot. This sequence of binary numbers is the adjacency matrix associated with the convolution block. More specifically, each sub-block contains the information of a column of the adjacency matrix. Figure 5 provides an example of the proposed encoding for $m = 5$. On the top of the figure, the encoding used in this study is shown. Below the chromosome, the corresponding adjacency matrix and table are displayed. It may be noticed that each block of the chromosome contains the columns of the adjacency matrix. At the bottom of Fig. 5, the corresponding network

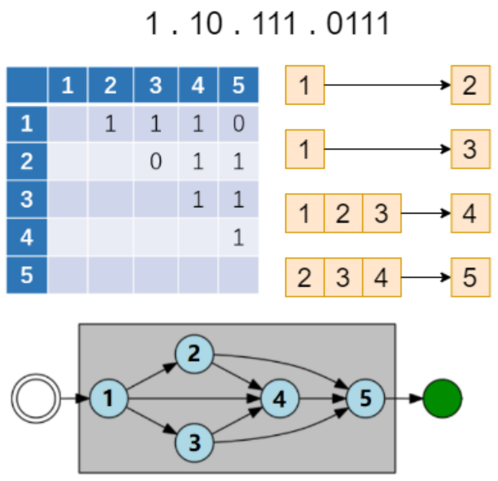


Fig. 5. Encoding of a convolution block CB_j (part of the chromosome) of the candidate CNN. The corresponding adjacency matrix and table are displayed as well as the graph of the encoded network. Convolution blocks formed by these binary blocks are the components of the CNN.

structure is represented. Also, like CB_j representing the connections in each block, the binary numbers in S is also the same representation as CB_j, and thus represents the connection between blocks.

The structure S is also a sequence of binary numbers which has $\frac{n(n-1)}{2}$ bits. The 1 indicates a link between two convolution blocks CB_i and CB_j while 0 indicates the dismiss of connections between blocks. The sequence S is also divided into sub-blocks composing the columns of the adjacency matrix that describes the topology of the interconnections among convolution blocks. The sequence P is composed of n binary numbers, one for each convolution block CB_j composing the CNN. The sequence P can be seen as a binary sequence Z where $z_j = 1$ represents the presence of pooling layers (P_j in Fig. 9) pointing to CB_{j+1} while $z_j = 0$ represents the absence of a pooling layer pointing to CB_{j+1}. The last binary number z_n indicates the presence or the absence of a pooling layer between CB_n and the fully connected layer FC.

Figure 6 provides the implementation details of the encoding mechanism in the context of the initialization of the population to be processed by MOGIG-Net.

The chromosome code only contains the topological structure before the fully connected layer. The connection mode between the components of each individual is determined at the beginning of the algorithm (residual connection[31] and close connection[33]).

```
Input: The limit of nodes number in each block ($M_{min}, M_{max}$), the limit of blocks number ($N_{min}, N_{max}$), the maximal pooling blocks number $K$.
Output: One chromosome

1: Generate a random number $n$, $n \in (N_{min}, N_{max})$;
2: $flag \leftarrow 0$
3: $gene \leftarrow$ empty string
4: while $flag < n$ do
5:   Generate a random number $m$, $m \in (M_{min}, M_{max})$;
6:   Generate a random sequence $s$ of $\frac{m(m-1)}{2}$ binary numbers and allocate them with "." separators in CB$_{flag}$
7:   Make sure that the sequence represents a connected graph, see Fig. 10.
8:   $gene \leftarrow gene + s + '-'$, the '-' in this paper is the separator between genes of blocks and pools
9:   $flag \leftarrow flag + 1$
10: end while
11: Generate a random sequence $s$ of $\frac{n(n-1)}{2}$ numbers and allocate them with "-" separators in S.
12: Make sure that the series can represent an oriented connected graph, see Fig. 10.
13: Generate a random sequence $p$ of $n$ binary numbers and allocate them in P
14: $gene \leftarrow gene + s + '-' + p$
    Return: chromosome
```

Fig. 6. MOGIG-Net encoding strategy and initialization.

Let us indicate with α the maximum number of cells of and with β the maximum number of blocks of the CNN. The longest possible code to search for contains L bits, and L is calculated by the following formula:

$$L = \frac{\beta \times \alpha \times (\alpha - 1)}{2} + \frac{\beta \times (\beta - 1)}{2} + \beta.$$

The search space contains up to 2^L possible candidate networks.

3.3. *Crossover and mutation*

Due to the encoding mechanism proposed in this paper, an ad-hoc crossover operator is here proposed to ensure that the offspring solutions meaningfully represent structures of neural networks.[13] Furthermore, a meaningful chromosome must represent a connected graph.

The proposed crossover operator combines two chromosomes I and II by selecting randomly some blocks from the first and then filling the missing gaps with the genotype of the second to ensure that the offspring is meaningful. Figure 7 provides the implementation details of the crossover.

For the chromosome I, two separators are randomly selected. Then the number of separators n between the two selected separators is calculated (line 6). Then, two separators in the chromosome II are selected while the number of separators between these two separators is ensured to be also n (line 7). Finally, the genes between the two separators are exchanged (line 8).

```
Input: Two parents, $p_1$ and $p_2$, probability of crossover $\mu \in (0, 1)$.
Output: Two offspring, $q_1$ and $q_2$.

1: Generate a random number $flag$;
2: if $flag > \mu$ then
3:   if num of separators in $p_1$ > num of separators in $p_2$ then
4:     $p_1, p_2 \leftarrow p_2, p_1$
5:   end if
6:   Select two different positions of separator randomly, $l_1$ and $l_2$, in $p_1$, (suppose $l_1 < l_2$);
7:   Select two different positions of separator randomly, $l_3$ and $l_4$, in $p_2$, and make sure that the num of separators in $p_1[l_1 : l_2]$ is the same as the num of separators in $p_2[l_3 : l_4]$;
8:   Exchange the parts $p_1[l_1 : l_2]$ and $p_2[l_3 : l_4]$, then get two offspring, $q_1$ and $q_2$;
9: else
10:   $q_1 \leftarrow p_1$;
11:   $q_2 \leftarrow p_2$;
12: end if
    Return: $q_1$ and $q_2$.
```

Fig. 7. MOGIG-Net crossover.

```
Input: One individual, $p$, probability of mutation $\nu \in (0, 1)$, bits to change, $n$.
Output: One individual, $q$.

1: Generate a random number $flag$;
2: if $flag > \nu$ then
3:   $q \leftarrow$ modify $n$ different bits in $q$;
4: else
5:   $q \leftarrow p$;
6: end if
   Return: $q$.
```

Fig. 8. MOGIG-Net mutation.

The mutation operation, outlined in Fig. 8, consists of the random flip from 0 to 1 or from 1 to 0 of a gene (except for the position of separator). Although the location of mutation changes is limited, the fact is that only small connection changes will affect all the input feature maps after this.

3.4. *Decoding of MOGIG-Net*

Figure 9 represents the construction of the CNN from its chromosome. At first, the CB$_j$ (in blue) are decoded. If the CB$_j$ is the same as that in the corresponding position in its parents, the module is copied from its parents. Otherwise, the module is generated according to the procedure illustrated in Fig. 5. Then, the S is decoded and the corresponding connection is represented by an input array of each block (such as the two red arrows pointing to CB$_3$). Finally, P is decoded and the corresponding position in each input array of each block is wrapped by an adaptive

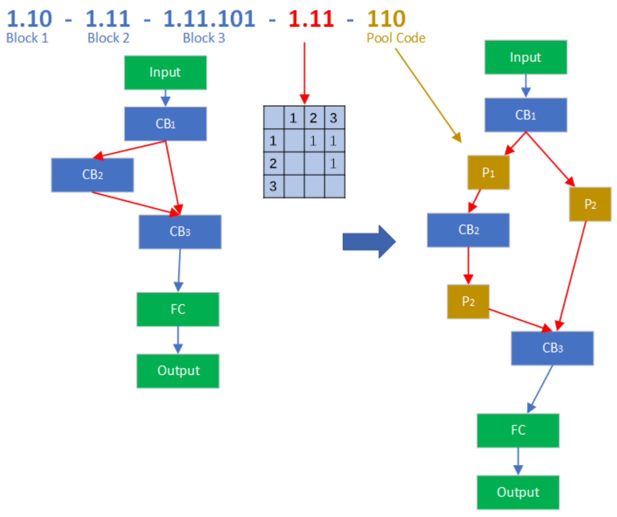
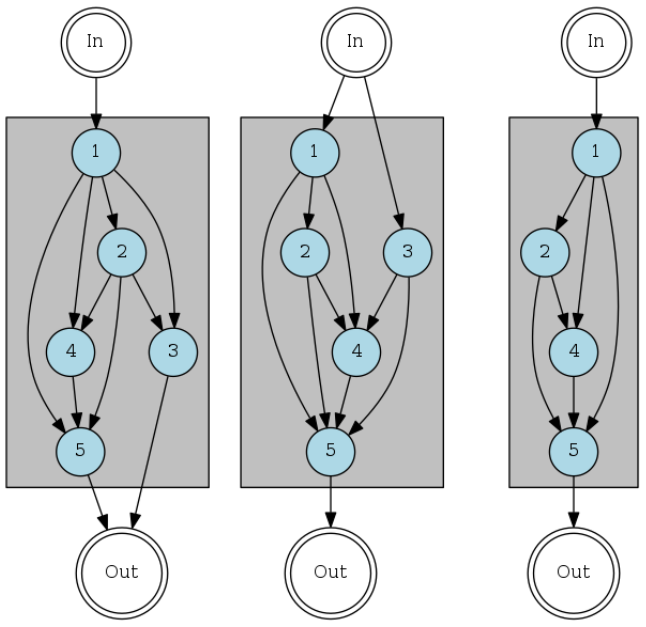


Fig. 9. (Color online) Construction of a CNN from its chromosome: The blocks or connections are decided by the part of encoding in the same color. The green squares represent fixed structures. FC means a fully connected layer. *P* means a pooling layer. Blocks are built in Fig. 5.



(a) (1.00.111.1111) (b) (1.11.110.1101) (c) (1.00.110.1101)

Fig. 10. Three scenarios to guarantee connected CNN blocks. In the left encoding, node 3 would have only inputs. Thus, an output link is generated to guarantee connectivity. In the central encoding, node 3 would have only outputs. Thus, an input link is generated to guarantee connectivity. In the right encoding, node 3 would be isolated. Thus, the node is removed from the graph.

pooling (like the right sub-figure in Fig. 9). The connection method in the detailed structure depends on the method which we choose before the algorithm. If we use the residual structure, we add the connection directly. If we use the dense structure, we adjust the channel and merge it by using the 1×1 convolution kernels to a unitize the channel number.

We also implemented a mechanism to handle missing connections within and among blocks when an adjacency list is generated. Let us consider at first the nodes within a block. If the generated solution contains a node which has inputs and no outputs, then a link between the node and the output node of the block is created. If the generated solution contains a node which has outputs and no outputs, then a link from the input node is generated. If a node has neither inputs nor outputs, then the node is removed. The same reasoning is performed about the connectivity among blocks where each node represents a block while input and output blocks of the CNN are considered instead of input and output nodes of the block. Figure 10 describes this mechanism by showing the three possible scenarios where node 3 has only inputs (left), has only outputs (center), has neither inputs nor outputs.

During the construction of a CNN from its chromosome, the skip connections (in blocks and between blocks), which need the sizes of the input and output to be the same, are fundamental to achieve a graph structure network. However, convolution and pooling operations can both change the size of the image. This characteristic of CNNs makes difficult to unify the input size of each part in the graph structure network. Therefore, we maintain the size consistency in the input and output of each block or search unit, so that the size reduction is completely controlled by the pooling layer. It is simple to keep the image size unchanged in the convolution block, only by adjusting the super parameters of the convolution kernel and avoiding the use of a pooling layer.

The formula for calculating the size of input and output is given in Eq. (1), where X_{out} and X_{in} are the size of the input and output, p is the number of padding around the input, d is the offset of two adjacent points of the dilated convolution, k is the size of the convolution kernel, and s is the step size of the convolution operation. Therefore, the size is controlled by means of the convolution kernel:

$$X_{\text{out}} = \left\lfloor \frac{X_{\text{in}} + 2 \times p - d \times (k-1) - 1}{s} + 1 \right\rfloor . \quad (1)$$

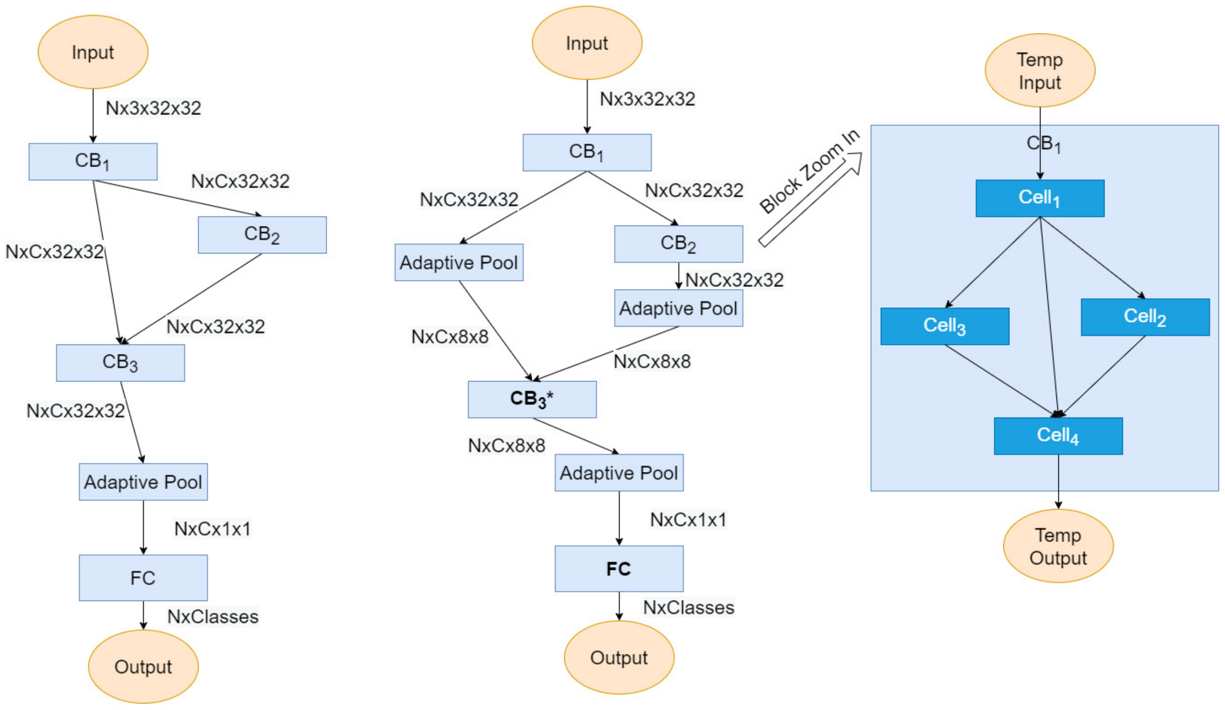


Fig. 11. The left subgraph is the macrostructure without pool layers. After executing line 11 of the algorithm in Fig. 6, the adaptive pooling is added at the specified location (center subgraph). The right subgraph is microstructure. Each block includes some convolution cells, and each cell is consist of 3×3 and 1×1 convolution kernels, which do not change the size of input.

Furthermore, since maintaining the consistency of image size outside the convolution block (i.e. the macrostructure) another countermeasure has been adopted. We also encode the reduced position of the size (but did not add into the genes) as the reduction of the size does not affect the use of convolution kernel, see Fig. 6, line 11.

We chose adaptive pooling, which is different from the traditional pooling operation. This operation can dynamically create pooled cores according to the input and output requirements, and it has been used in the last layer of many existing models.[31,33] The step size of the adaptive pooling layer can be obtained by the following equation:

$$\text{stride} = \left\lfloor \frac{\text{size}_{\text{in}}}{\text{size}_{\text{out}}} \right\rfloor, \tag{2}$$

where size_{in} is the size of input feature map and size_{out} is the size of output feature map. The size of pool $\text{size}_{\text{pool}}$ is then calculated in the following equation:

$$\text{size}_{\text{pool}} = \text{size}_{\text{in}} - \text{stride} * (\text{size}_{\text{out}} - 1). \tag{3}$$

On the basis of these two formulas, we can adjust the kernel of the adaptive pooling layer from the size of the input feature map and the size of the desired output feature map. As shown in Fig. 11, the two pool layers before the block and FC marked in bold because we choose to add adaptive pools before them. In this way, we can control the size of input and output in each layer by controlling the position of adaptive pooling. The location of adaptive pooling and the combination of these channels are referred to as the detailed structure of the individual and are recorded separately.

3.5. *Evaluation and environment selection*

We divide the training sets D into two parts, 80% of which are real training sets D_{train}, and the rest are validation sets D_{valid}. When the new population of offspring solutions is generated, their performance must be assessed to select the population undergoing the following generation. The networks composing the new population undergo training by means of the training set D_{train}. When the change range is below a pre-arranged threshold, the learning rate is adjusted accordingly. If the learning rate adjustment is less than a prearranged value, the training will be stopped.

In our approach, we use weight inheritance to speed up the search. Since our crossover operation can ensure that most of the modules of the network remain unchanged, the weight of the model constructed by the child will directly inherit the weight from the model of the parent. This method, like weight sharing, can make the network model obtain a relatively high accuracy rate at the early stage of evolution. In this way, we only need to continue training at a relatively small learning rate to achieve the best performance of each network.

After the training, the accuracy $q.acc$ (that is the error rate) of the network is assessed by means of the validation set D_{valid}. Furthermore, the model size in terms of the number of parameters $q.params$ is also calculated. Both the scores $q.acc$ and $q.params$ characterize the quality of the candidate CNN. The nondominated sorting[50] is used to select among parent and offspring solutions the population undergoing the following generation, which often used to evaluate the quality of two solutions in the process of multi-objective optimization.[93] The condition for one individual to dominate another is to have a performance not worse than the other according to all objective and to outperform it according to at least one objective.

Figure 12 provides the implementation details of evaluation and selection mechanisms.

```
Input: The population $P_t$, the training set $D_{train}$, the validation set $D_{valid}$
Output: The new population $P_{t+1}$

for all individual $q$ in population $P$ do
  Check the database of fingerprint
  if fingerprint of $q$ is in the database then
    Get $q.acc$ and $q.params$ from database;
  else
    $cnn \leftarrow$ Generate the network with $q$;
    train $cnn$ on $D_{train}$ until the loss and accuracy don't change significantly;
    $q.acc \leftarrow$ the rate of accuracy assessed on the valid set;
    $q.params \leftarrow$ the number of parameters contained in the model $cnn$ itself;
  end if
  Update individual $q$ in population $P$;
end for
Do non-dominated sorting [50] and select half of the individuals who were better at multiple goals from $P_{t+1}$.
Return: $P_{t+1}$
```

Fig. 12. MOGIG-Net multi-objective evaluation and selection.

3.6. *Limitations of MOGIG-Net and countermeasures*

Without a prior knowledge on the problem, each connection has initially the same probability to be set as 0 or as 1. Thus, on average initialized solutions contain approximately half of the skip connections, many of them being unnecessary. These skip connections can cause a slow down of the network training. Thus, the search efficiency of our method is rather low in the early stages. However, the method of weight inheritance accelerates the search and partially mitigates this limitation. Already from the second generation of the population, we observed a large number of excellent structures in the population and its parameters are retained along with the encoding, which makes the training process overall efficient and yields high-performance candidate solutions.

4. Experiments

This section displays the results of the proposed MOGIG-Net on two popular datasets and compares its performance with that of 17 NAS methods previously proposed in the literature.

The popular datasets considered in this study are Cifar-10 and Cifar-100 proposed by the Canadian Institute for Advanced Research.[37] These two datasets are often used to verify the performance of network models. Each dataset comprises 60,000 images, including 50,000 in the training set and 10,000 in the test set. Each image is a three-channel color image, and the height and the width are both 32. There are 10 categories in Cifar-10 and 100 categories in Cifar-100. Both Cifar-10 and Cifar-100 come from a larger dataset of 80 million small images. Therefore, to a certain extent, Cifar-10 and Cifar-100 can illustrate the predictive ability of the model.

Table 1 displays the results of MOGIG-Net and 21 NAS competitors on Cifar-10 and Cifar-100. The listed methods are divided into three design categories: NAS human design, single-objective approaches and multi-objective approaches. For each NAS method considered in this study, the reference to its original implementation. For each method we report the result of the objectives in the proposed model, that is the accuracy $q.acc$ expressed in terms to percentage error for Cifar-10 and Cifar-100 and the complexity $q.param$ expressed in million of parameters of the network designed by the

Table 1. Results on Cifar-10 and Cifar-100 datasets[37] of the proposed MOGIG-Net against 21 NAS methods. The percentage error "Error Rate (%)" and number of parameters expressed in million pf parameters "Params (M)" are reported.

Name	Params (M)	Error rate (%)	
		Cifar-10	Cifar-100
Human design			
DenseNet ($k = 12$)[33]	1.0	5.24	24.42
ResNet (depth = 101)[31]	1.7	6.43	25.16
ResNet (depth = 1202)[31]	10.2	7.93	27.82
MobileNetV2[68]	2.2	4.26	19.20
NASNet-A Mobile[99]	4.2	3.17	16.10
EfficientNet-B0[86]	4.0	1.90	11.90
MixNet[87]	3.5	2.08	—
DARTS[47]	3.4	2.83	—
VGG[74]	20.1	6.66	28.05
NIN[44]	—	8.81	35.68
Single-objective approaches			
Genetic CNN[91]	—	7.10	29.05
Block-QNN[97]	39.8	3.50	—
Block-QNN-s[97]	6.1	4.38	20.65
LaNet-S[90]	3.2	1.63	—
LaNet-L[90]	44.1	0.99	—
Oneshot-LaNet-S[90]	3.6	1.68	—
Oneshot-LaNet-L[90]	45.3	1.20	—
Large-scale evolution[63]	5.4	5.40	
	40.4		23.00
MetaQNN[9]	—	6.92	27.14
AE-CNN[80]	2.0	4.30	
	5.4		20.85
Multi-objective approaches			
NSGA-Net[50]	0.2	4.67	
	4.0	2.02	
	0.2		25.17
	4.1		14.38
MOGIG-Net	0.9	4.67	—
	3.0	3.13	—
	3.7	2.01	—
	0.7	—	24.71
	3.2	—	18.23
	3.7	—	14.38

corresponding NAS method. We may observe that the proposed MOGIG-Net can efficiently detect networks which combine a relatively low number of parameters and a low percentage error. For example, none of the 17 competitor NAS methods can achieve an error rate of 14.38% on Cifar-100 with only 3.7 million parameters. With respect to NSGA-Net,[50] that is a recent NAS method considered the state-of-the-art in the field, the proposed MOGIG-Net designed networks with a comparable performance notwithstanding a lower number of parameters (approximately 10% fewer parameters).

Figures 13 and 14 display the solutions in the objective space considered in this study detected by the proposed MOGIG-Net and its competitor. To enhance the readability of the figures, we present a zoom around the nondominated solutions.

We noticed that when the network structure is relatively large, the number of pooling in the detailed structure greatly affects the required training time and the memory space. When the number of pooling is small and the network structure is large, the size of intermediate variables is very large and the training time is very long. The results in this study have been detected after two weeks of calculation.

Experimental results show that for networks with similar structures, the accuracy of large models is higher than that of small models, including our method. The reason of this phenomenon is that the increase in the number of parameters appears to improve the generalization capability of the model. Therefore, the maximum accuracy that can be achieved with large models is higher than that of smaller models.

The results in Figs. 13 and 14 show that MixNet and MobileNetV2 display excellent performance. However, MixNet and MobileNetV2, unlike the proposed MOGIG-Net are human-designed networks with a predefined purpose. Thus, the performance of the methods cannot be directly compared. Also, LaNet produced a solution that dominates the MOGIG-Net solution for Cifar-10. We suspect that this may be because LaNet tends to select large models with high accuracy, and we come to this conclusion because some of the networks in the search

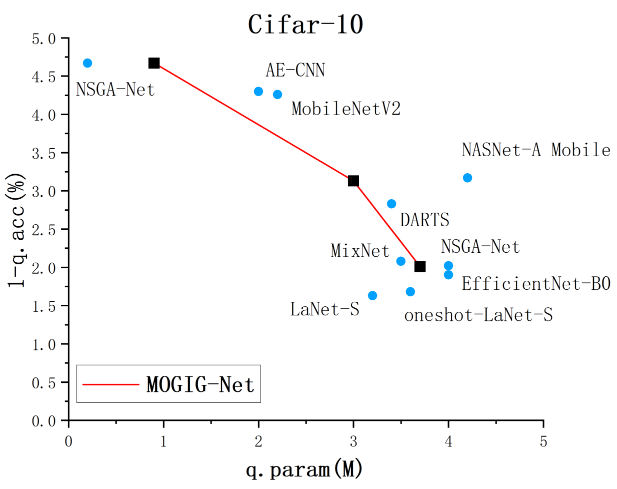


Fig. 13. Solutions detected by MOGIG-Net and its competitors represented in the objective space (Cifar-10).

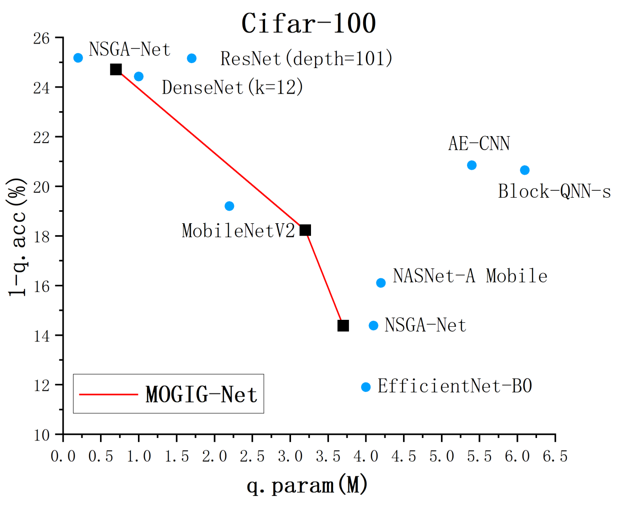


Fig. 14. Solutions detected by MOGIG-Net and its competitors represented in the objective space (Cifar-100).

space, like LaNet-L and oneshot-LaNet-L, seems to be large.

However, when the network structure is relatively small, with the increase of total computation times (Multiply–Adds operations), the generalization performance of the network is also improving. Consequently, the next step of this work will be to transform the network structure and/or to determine the number of pooling which is randomly added to the network structure, instead of randomly generating several pooling layers and inserting them into random locations.

Since numerical results indicate that the proposed MOGIG-Net is able to design excellent CNNs, a future direction of our research will include the extension of the encoding strategy to other ingenious neural systems recently proposed in the literature, such as Enhanced Probabilistic Neural Network,[5] Neural Dynamic Classification Algorithm,[61] Dynamic Ensemble Learning Algorithm,[6] and Finite Element Machine for Fast Learning[60]

5. Conclusion

This paper proposes a NAS method to design CNNs with high performance in terms of accuracy and a limited impact on the computational resources.

The proposed algorithm indicated with MOGIG-Net makes use of a novel block logic based on adjacency list to compose the network structure. The encoding mechanism proposed in this paper can naturally represent the structure of any graph. Moreover, MOGIG-Net employs ad-hoc crossover and mutation operators which are designed to explore the search space and identify potential candidate structures. At last, the proposed network encoding enables that the parent structures can be effectively and naturally transferred to the offspring during the crossover process. The proposed approach overcomes the limitation of classical NAS approaches based on EAs which require a search in a large space and an overhead due to multiple re-training sessions. Numerical results on two popular datasets Cifar-10 and Cifar-100 show that MOGIG-Net can exceed most existing network structures.

This paper confirms that multi-objective optimization modeling is a promising direction of research in the field of NAS. Future research will consider further objectives and strategies to reduce the computational cost of the training by e.g. limiting the number of skip connections in the first generation.

Acknowledgments

Yu Xue and Pengcheng Jiang contributed equally to this work and should be considered *co-first* authors.

This work was partially supported by the National Natural Science Foundation of China (61876089, 61876185, and 61902281), the opening Project of Jiangsu Key Laboratory of Data Science and Smart Software (No. 2019DS301), the Natural Science Foundation of Jiangsu Province (BK20141005), the Natural Science Foundation of the Jiangsu Higher Education Institutions of China (14KJB520025).

References

1. H. A. Abbass, Pareto neuro-evolution: Constructing ensemble of neural networks using multi-objective optimization, in *IEEE Congress on Evolutionary Computation*, Vol. **3** (IEEE, 2003), pp. 2074–2080.
2. U. R. Acharya, S. L. Oh, Y. Hagiwara, J. H. Tan and H. Adeli, Deep convolutional neural network for the automated detection and diagnosis of seizure using EEG signals, *Comput. Biol. Med.* **100** (2018) 270–278.
3. U. R. Acharya, S. L. Oh, Y. Hagiwara, J. H. Tan, H. Adeli and D. P. Subha, Automated EEG-based screening of depression using deep convolutional neural network, *Comput. Methods Programs Biomed.* **161** (2018) 103–113.
4. A. Agogino, K. Stanley and R. Miikkulainen, Online interactive neuro-evolution, *Neural Process. Lett.* **11**(1) (2000) 29–38.
5. M. Ahmadlou and H. Adeli, Enhanced probabilistic neural network with local decision circles: A robust classifier, *Integr. Comput.-Aided Eng.* **17**(3) (2010) 197–210.
6. K. M. R. Alam, N. Siddique and H. Adeli, A dynamic ensemble learning algorithm for neural networks, *Neural Comput. Appl.* **32**(12) (2020) 8675–8690.
7. A. H. Ansari, P. J. Cherian, A. Caicedo, G. Naulaers, M. De Vos and S. Van Huffel, Neonatal seizure detection using deep convolutional neural networks, *Int. J. Neural Syst.* **29**(04) (2019) 1850011.
8. A. Asseman, N. Antoine and A. S. Ozcan, Accelerating deep neuroevolution on distributed FPGAs for reinforcement learning problems, *ACM J. Emerg. Technol. Comput. Syst.* **17**(2) (2021) 1–17.
9. B. Baker, O. Gupta, N. Naik and R. Raskar, Designing neural network architectures using reinforcement learning, in *Int. Conf. Learning Representations* (Toulon, France, 2017), p. 18.
10. B. Baker, O. Gupta, R. Raskar and N. Naik, Accelerating neural architecture search using performance prediction, in *Int. Conf. Learning Representations* (Vancouver, BC, Canada, 2018).
11. G. Bender, P.-J. Kindermans, B. Zoph, V. Vasudevan and Q. Le, Understanding and simplifying one-shot architecture search, in *Int. Conf. Machine Learning* (Stockholm, Sweden, 2018), pp. 550–559.
12. S. Bhat, U. R. Acharya, Y. Hagiwara, N. Dadmehr and H. Adeli, Parkinson's disease: Cause factors, measurable indicators, and early diagnosis, *Comput. Biol. Med.* **102** (2018) 234–241.
13. C. Blum, R. Chiong, M. Clerc, K. De Jong, Z. Michalewicz, F. Neri and T. Weise, Evolutionary optimization, in *Variants of Evolutionary Algorithms for Real-world Applications* (Springer, 2012), pp. 1–29.
14. H. Cai, J. Yang, W. Zhang, S. Han and Y. Yu, Path-level network transformation for efficient architecture search, in *PMLR Int. Conf. Machine Learning* (Stockholm, Sweden, 2018), pp. 678–687.
15. H. Cai, L. Zhu and S. Han, ProxylessNAS: Direct neural architecture search on target task and hardware, in *Int. Conf. Learning Representations* (Vancouver, BC, Canada, 2018), p. 13.
16. X. Cao, J. Yao, Z. Xu and D. Meng, Hyperspectral image classification with convolutional neural network and active learning, *IEEE Trans. Geosci. Remote Sens.* **58**(7) (2020) 4604–4616.
17. F. Charte, A. J. Rivera, F. Martínez and M. J. del Jesus, EvoAAA: An evolutionary methodology for automated neural autoencoder architecture search, *Integr. Comput.-Aided Eng.* **27**(3) (2020) 211–231.

18. L.-C. Chen, M. Collins, Y. Zhu, G. Papandreou, B. Zoph, F. Schroff, H. Adam and J. Shlens, Searching for efficient multi-scale architectures for dense image prediction, in *Advances in Neural Information Processing Systems* (Montréal, Canada, 2018), pp. 8699–8710.
19. F. Chollet, Xception: Deep learning with depthwise separable convolutions, in *Proc. IEEE Conf. Computer Vision and Pattern Recognition* (Honolulu, HI, USA, 2017), pp. 1251–1258.
20. X. Dai *et al.*, ChamNet: Towards efficient network design through platform-aware model adaptation, in *Proc. IEEE Conf. Computer Vision and Pattern Recognition* (Long Beach, CA, USA, 2019), pp. 11398–11407.
21. A. Diba, A. Pazandeh and L. Van Gool, Efficient two-stream motion and appearance 3D CNNs for video classification, in *Proc. European Conf. Computer Vision* (*ECCV*) (Amsterdam, Netherlands, 2016), pp. 1–4.
22. X. Dong and Y. Yang, One-shot neural architecture search via self-evaluated template network, in *Proc. IEEE/CVF Int. Conf. Computer Vision* (Seoul, Korea, 2019), pp. 3681–3690.
23. T. Elsken, J. H. Metzen and F. Hutter, Efficient multi-objective neural architecture search via Lamarckian evolution, in *Int. Conf. Learning Representations* (Vancouver, BC, Canada, 2018), p. 23.
24. T. Elsken, J.-H. Metzen and F. Hutter, Simple and efficient architecture search for convolutional neural networks, in *Int. Conf. Learning Representations* (Vancouver, BC, Canada, 2018), p. 14.
25. J. Fang, Y. Sun, Q. Zhang, Y. Li, W. Liu and X. Wang, Densely connected search space for more flexible neural architecture search, in *Proc. IEEE/CVF Conf. Computer Vision and Pattern Recognition* (Seattle, WA, USA, 2020), pp. 10628–10637.
26. F. Gomez, J. Schmidhuber and R. Miikkulainen, Efficient non-linear control through neuroevolution, in *European Conf. Machine Learning* (Springer, 2006), pp. 654–662.
27. F. J. Gomez and R. Miikkulainen, Solving non-Markovian control tasks with neuroevolution, in *Proc. Int. Joint Conf. Artificial Intelligence*, Vol. 99, Stockholm, Sweden, 1999, pp. 1356–1361.
28. X. Gong, S. Chang, Y. Jiang and Z. Wang, AutoGAN: Neural architecture search for generative adversarial networks, in *Proc. IEEE Int. Conf. Computer Vision* (Long Beach, CA, USA, 2019), pp. 3224–3234.
29. Z. Guo, X. Zhang, H. Mu, W. Heng, Z. Liu, Y. Wei and J. Sun, Single path one-shot neural architecture search with uniform sampling, in *European Conf. Computer Vision* (Springer, 2020), pp. 544–560.
30. M. Hausknecht, J. Lehman, R. Miikkulainen and P. Stone, A neuroevolution approach to general Atari game playing, *IEEE Trans. Comput. Intell. AI Games* **6**(4) (2014) 355–366.
31. K. He, X. Zhang, S. Ren and J. Sun, Deep residual learning for image recognition, in *Proc. IEEE Conf. Computer Vision and Pattern Recognition* (Las Vegas, NV, USA, 2016), pp. 770–778.
32. O. M. Hooman, M. M. Al-Rifaie and M. A. Nicolaou, Deep neuroevolution: Training deep neural networks for false alarm detection in intensive care units, in *2018 26th IEEE European Signal Processing Conf.* (*EUSIPCO*) (Rome, Italy, 2018), pp. 1157–1161.
33. G. Huang, Z. Liu, L. Van Der Maaten and K. Q. Weinberger, Densely connected convolutional networks, in *Proc. IEEE Conf. Computer Vision and Pattern Recognition* (Honolulu, HI, USA, 2017), pp. 4700–4708.
34. H. Jiang, F. Gao, X. Xu, F. Huang and S. Zhu, Attentive and ensemble 3D dual path networks for pulmonary nodules classification, *Neurocomputing* **398** (2020) 422–430.
35. N. Karthikeyan and R. Sukanesh, Cloud based emergency health care information service in India, *J. Med. Syst.* **36**(6) (2012) 4031–4036.
36. A. Krizhevsky, One weird trick for parallelizing convolutional neural networks, tech. report (2014).
37. A. Krizhevsky *et al.*, Learning multiple layers of features from tiny images, tech. report, University of Toronto (2009).
38. H. Kwasnicka and M. Paradowski, Efficiency aspects of neural network architecture evolution using direct and indirect encoding, in *Adaptive and Natural Computing Algorithms* (Springer, 2005), pp. 405–408.
39. S. Li, X. Zhao and G. Zhou, Automatic pixel-level multiple damage detection of concrete structure using fully convolutional network, *Comput.Aided Civ. Infrastruct. Eng.* **34**(7) (2019) 616–634.
40. X. Li, Y. Jiang, M. Li and S. Yin, Lightweight attention convolutional neural network for retinal vessel image segmentation, *IEEE Trans. Ind. Inf.* **17**(3) (2020) 1958–1967.
41. X. Li, C. Lin, C. Li, M. Sun, W. Wu, J. Yan and W. Ouyang, Improving one-shot NAS by suppressing the posterior fading, in *Proc. IEEE/CVF Conf. Computer Vision and Pattern Recognition* (Seattle, WA, USA, 2020), pp. 13836–13845.
42. Y. Li, Z. Yu, Y. Chen, C. Yang, Y. Li, X. Allen Li and B. Li, Automatic seizure detection using fully convolutional nested LSTM, *Int. J. Neural Syst.* **30**(04) (2020) 2050019.
43. L.-C. Lin, C.-S. Ouyang, R.-C. Wu, R.-C. Yang and C.-T. Chiang, Alternative diagnosis of epilepsy in children without epileptiform discharges using deep convolutional neural networks, *Int. J. Neural Syst.* **30**(5) (2020) 1850060.
44. M. Lin, Q. Chen and S. Yan, Network in network, in *Int. Conf. Learning Representations* (Banff, Canada, 2014), p. 10.

45. C. Liu, B. Zoph, M. Neumann, J. Shlens, W. Hua, L.-J. Li, L. Fei-Fei, A. Yuille, J. Huang and K. Murphy, Progressive neural architecture search, in *Proc. European Conf. Computer Vision* (*ECCV*) (Munich, Germany, 2018), pp. 19–34.
46. G. Liu, W. Zhou and M. Geng, Automatic seizure detection based on S-transform and deep convolutional neural network, *Int. J. Neural Syst.* **30**(04) (2020) 1950024.
47. H. Liu, K. Simonyan and Y. Yang, DARTS: Differentiable architecture search, in *Int. Conf. Learning Representations* (Vancouver, BC, Canada, 2018), p. 13.
48. Z. Liu, Z. Wu, T. Li, J. Li and C. Shen, GMM and CNN hybrid method for short utterance speaker recognition, *IEEE Trans. Ind. Inf.* **14**(7) (2018) 3244–3252.
49. Z. Lu, K. Deb and V. N. Boddeti, MUXConv: Information multiplexing in convolutional neural networks, in *Proc. IEEE/CVF Conf. Computer Vision and Pattern Recognition* (Seattle, WA, USA, 2020), pp. 12044–12053.
50. Z. Lu, I. Whalen, V. Boddeti, Y. Dhebar, K. Deb, E. Goodman and W. Banzhaf, NSGA-Net: Neural architecture search using multi-objective genetic algorithm, in *Proc. Genetic and Evolutionary Computation Conf.* (Prague, Czech Republic, 2019), pp. 419–427.
51. N. Ma, X. Zhang, H.-T. Zheng and J. Sun, ShuffleNet V2: Practical guidelines for efficient CNN architecture design, in *Proc. European Conf. Computer Vision* (*ECCV*) (Munich, Germany, 2018), pp. 116–131.
52. Z. Ma, Reachability analysis of neural masses and seizure control based on combination convolutional neural network, *Int. J. Neural Syst.* **30**(1) (2020) 1950023.
53. K. Maeda, S. Takahashi, T. Ogawa and M. Haseyama, Convolutional sparse coding-based deep random vector functional link network for distress classification of road structures, *Comput.-Aided Civ. Infrastruct. Eng.* **34**(8) (2019) 654–676.
54. O. M. Manzanera, S. K. Meles, K. L. Leenders, R. J. Renken, M. Pagani, D. Arnaldi, F. Nobili, J. Obeso, M. R. Oroz, S. Morbelli and N. M. Mauritis, Scaled subprofile modeling and convolutional neural networks for the identification of Parkinsons disease in 3D nuclear imaging data, *Int. J. Neural Syst.* **29**(9) (2019) 1950010.
55. H. Mo, L. L. Custode and G. Iacca, Evolutionary neural architecture search for remaining useful life prediction, *Appl. Soft Comput.* **108** (2021) 107474.
56. H. Mo, F. Lucca, J. Malacarne and G. Iacca, Multi-head CNN-LSTM with prediction error analysis for remaining useful life prediction, in *2020 27th Conf. Open Innovations Association* (*FRUCT*) (IEEE, 2020), pp. 164–171.
57. R. Negrinho, M. Gormley, G. J. Gordon, D. Patil, N. Le and D. Ferreira, Towards modular and programmable architecture search, in *Advances in Neural Information Processing Systems* (Vancouver, Canada, 2019), pp. 13715–13725.
58. F. Neri, *Linear Algebra for Computational Sciences and Engineering*, 2nd edn. (Springer, 2019).
59. B. K. Oh, B. Glisic, Y. Kim and H. S. Park, Convolutional neural network based wind induced response estimation model for tall buildings, *Comput.-Aided Civ. Infrastruct. Eng.* **34**(10) (2019) 843–858.
60. D. R. Pereira, M. A. Piteri, A. N. Souza, J. P. Papa and H. Adeli, FEMa: A finite element machine for fast learning, *Neural Comput. Appl.* **32**(10) (2020) 6393–6404.
61. M. H. Rafiei and H. Adeli, A new neural dynamic classification algorithm, *IEEE Trans. Neural Netw. Learn. Syst.* **28**(12) (2017) 3074–3083.
62. E. Real, A. Aggarwal, Y. Huang and Q. V. Le, Regularized evolution for image classifier architecture search, in *Proc. AAAI Conf. Artificial Intelligence*, Vol. 33, No. 01 (Honolulu, Hawaii, USA, 2019), pp. 4780–4789.
63. E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. V. Le and A. Kurakin, Large-scale evolution of image classifiers, in *Int. Conf. Machine Learning* (Sydney, Australia, 2017), pp. 2902–2911.
64. S. Risi and K. O. Stanley, Deep neuroevolution of recurrent and discrete world models, in *Proc. Genetic and Evolutionary Computation Conf.* (Prague, Czech Republic, 2019), pp. 456–462.
65. S. Rostami, F. Neri and M. Epitropakis, Progressive preference articulation for decision making in multi-objective optimisation problems, *Integr. Comput.-Aided Eng.* **24**(4) (2017) 315–335.
66. S. Rostami, F. Neri and K. Gyaurski, On algorithmic descriptions and software implementations for multi-objective optimisation: A comparative study, *SN Comput. Sci.* **1**(5) (2020) 1–23.
67. C. Saltori, S. Roy, N. Sebe and G. Iacca, Regularized evolutionary algorithm for dynamic neural topology search, in *Int. Conf. Image Analysis and Processing* (Springer, 2019), pp. 219–230.
68. M. Sandler, A. Howard, M. Zhu, A. Zhmoginov and L.-C. Chen, MobileNetV2: Inverted residuals and linear bottlenecks, in *Proc. IEEE Conf. Computer Vision and Pattern Recognition* (Salt Lake City, UT, USA, 2018), pp. 4510–4520.
69. K. C. Sarma and H. Adeli, Fuzzy discrete multicriteria cost optimization of steel structures, *J. Struct. Eng.* **126**(11) (2000) 1339–1347.
70. K. C. Sarma and H. Adeli, Bilevel parallel genetic algorithms for optimization of large steel structures, *Comput.-Aided Civ. Infrastruct. Eng.* **16**(5) (2001) 295–304.
71. W. W. Seeley, J. M. Allman, D. A. Carlin, R. K. Crawford, M. N. Macedo, M. D. Greicius, S. J.

Dearmond and B. L. Miller, Divergent social functioning in behavioral variant frontotemporal dementia and Alzheimer disease: Reciprocal networks and neuronal evolution, *Alzheimer Dis. Assoc. Disord.* **21**(4) (2007) S50–S57.

72. J. Shen, X. Xiong, Z. Xue and Y. Bian, A convolutional neural-network-based pedestrian counting model for various crowded scenes, *Comput.-Aided Civ. Infrastruct. Eng.* **34**(10) (2019) 897–914.
73. W. Shin, S.-J. Bu and S.-B. Cho, 3D-convolutional neural network with generative adversarial network and autoencoder for robust anomaly detection in video surveillance, *Int. J. Neural Syst.* **30**(6) (2020) 2050034.
74. K. Simonyan and A. Zisserman, Very deep convolutional networks for large-scale image recognition, in *Int. Conf. Learning Representations* (San Diego, CA, USA, 2015), p. 14.
75. M. G. Soto and H. Adeli, Many-objective control optimization of high-rise building structures using replicator dynamics and neural dynamics model, *Struct. Multidiscip. Optim.* **56**(6) (2017) 1521–1537.
76. K. O. Stanley, J. Clune, J. Lehman and R. Miikkulainen, Designing neural networks through neuroevolution, *Nat. Mach. Intell.* **1**(1) (2019) 24–35.
77. F. P. Such, V. Madhavan, E. Conti, J. Lehman, K. O. Stanley and J. Clune, Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning, in *Int. Conf. Learning Representations* (New Orleans, USA, 2019), p. 16.
78. Y. Sun, H. Wang, B. Xue, Y. Jin, G. G. Yen and M. Zhang, Surrogate-assisted evolutionary deep learning using an end-to-end random forest-based performance predictor, *IEEE Trans. Evol. Comput.* **24**(2) (2019) 350–364.
79. Y. Sun, B. Xue, M. Zhang and G. G. Yen, A particle swarm optimization-based flexible convolutional autoencoder for image classification, *IEEE Trans. Neural Netw. Learn. Syst.* **30**(8) (2018) 2295–2309.
80. Y. Sun, B. Xue, M. Zhang and G. G. Yen, Completely automated CNN architecture design based on blocks, *IEEE Trans. Neural Netw. Learn. Syst.* **31**(4) (2019) 1242–1254.
81. Y. Sun, B. Xue, M. Zhang and G. G. Yen, Evolving deep convolutional neural networks for image classification, *IEEE Trans. Evol. Comput.* **24**(2) (2019) 394–407.
82. Y. Sun, B. Xue, M. Zhang, G. G. Yen and J. Lv, Automatically designing CNN architectures using the genetic algorithm for image classification, *IEEE Trans. Cybern.* **50** (2020) 3840–3854.
83. C. Szegedy, S. Ioffe, V. Vanhoucke and A. A. Alemi, Inception-v4, inception-ResNet and the impact of residual connections on learning, in *Proc. Thirty-First AAAI Conf. Artificial Intelligence* (California, USA, 2017), pp. 4278–4284.
84. C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke and A. Rabinovich, Going deeper with convolutions, in *Proc. IEEE Conf. Computer Vision and Pattern Recognition* (Boston, MA, USA, 2015), pp. 1–9.
85. M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard and Q. V. Le, MnasNet: Platform-aware neural architecture search for mobile, in *Proc. IEEE Conf. Computer Vision and Pattern Recognition* (Long Beach, CA, USA, 2019), pp. 2820–2828.
86. M. Tan and Q. Le, EfficientNet: Rethinking model scaling for convolutional neural networks, in *Int. Conf. Machine Learning* (Long Beach, CA, USA, 2019), pp. 6105–6114.
87. M. Tan and Q. V. Le, MixConv: Mixed depthwise convolutional kernels, in *30th British Machine Vision Conf. 2019* (*BMVC 2019*), Cardiff, UK (BMVA Press, 2019), p. 74.
88. K. Thurnhofer-Hemsi, E. López-Rubio, N. Roe-Vellve and M. A. Molina-Cabello, Multiobjective optimization of deep neural networks with combinations of Lp-norm cost functions for 3D medical image super-resolution, *Integr. Comput.-Aided Eng.* **27**(1) (2020) 1–19.
89. B. Wang, Y. Sun, B. Xue and M. Zhang, Evolving deep neural networks by multi-objective particle swarm optimization for image classification, in *Proc. Genetic and Evolutionary Computation Conf.* (Prague, Czech Republic, 2019), pp. 490–498.
90. L. Wang, S. Xie, T. Li, R. Fonseca and Y. Tian, Sample-efficient neural architecture search by learning actions for Monte Carlo tree search, to appear in *IEEE Trans. Pattern Anal. Mach. Intell.* (2021) 14.
91. L. Xie and A. Yuille, Genetic CNN, in *Proc. IEEE Int. Conf. Computer Vision* (Venice, Italy, 2017), pp. 1379–1388.
92. S. Xie, H. Zheng, C. Liu and L. Lin, SNAS: Stochastic neural architecture search, in *Int. Conf. Learning Representations* (Vancouver, BC, Canada, 2018), p. 17.
93. Y. Xue, Y. Tang, X. Xu, J. Liang and F. Neri, Multi-objective feature selection with missing data in classification, *IEEE Trans. Emerg. Top. Comput. Intell.* (2021) 10.
94. X. Yao and M. M. Islam, Evolving artificial neural network ensembles, *IEEE Comput. Intell. Mag.* **3**(1) (2008) 31–42.
95. Y. Zhang, Y. Miyamori, S. Mikami and T. Saito, Vibration based structural state identification by a 1-dimensional convolutional neural network, *Comput. Aided Civ. Infrastruct. Eng.* **34**(9) (2019) 822–839.
96. D. Zhipeng, W. Jingcheng, X. Yumin, M. Qingmin and W. Xiaoming, Voiceprint recognition based on BP neural network and CNN, *J. Phys., Conf. Ser.* **1237**(3) (2019) 032032.

97. Z. Zhong, J. Yan and C.-L. Liu, Practical network blocks design with q-learning, arXiv:1708.05552.
98. B. Zoph and Q. V. Le, Neural architecture search with reinforcement learning, in *Int. Conf. Learning Representations* (Toulon, France, 2017), p. 16.
99. B. Zoph, V. Vasudevan, J. Shlens and Q. V. Le, Learning transferable architectures for scalable image recognition, in *Proc. IEEE Conf. Computer Vision and Pattern Recognition* (Salt Lake City, UT, USA, 2018), pp. 8697–8710.